5 Видео и аудио

Воспроизведение видео с помощью HTML5

Одна из наиболее часто упоминаемых возможностей HTML5 — обработка видео. Однако эта шумиха связана не с новыми инструментами, предоставляемыми HTML5 для этой цели, а с тем фактом, что в момент превращения видео в центральный элемент Интернета все автоматически предположили, что теперь следует ожидать его встроенной поддержки в любых браузерах. Создавалось впечатление, что значимость видео осознавали все, кроме разработчиков технологий для Сети.

Но когда встроенная поддержка и даже стандарт, позволяющий создавать кросс-браузерные приложения для обработки видео, появились, выяснилось, что все гораздо сложнее, чем ожидалось. Причины отказаться от реализации видео оказывались намного серьезнее, чем разработка необходимых для этого кодеков.

Но несмотря на все сложности создатели HTML5 наконец представили элемент, предназначенный для вставки видеофайлов в документы HTML и их воспроизведения. Элементу `<video>` для выполнения своих функций требуются лишь открывающий и закрывающий теги и несколько несложных параметров. Синтаксис чрезвычайно прост, а обязательным является только атрибут `src`.

Листинг 5.1. Базовый синтаксис элемента <video>

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <title>Видеопроигрыватель</title>
</head>
<body>
<section id="player">
  <video src="http://minkbooks.com/content/trailer.mp4" controls>
  </video>
</section>
</body>
</html>
```

Теоретически кода из листинга 5.1 должно быть достаточно — подчеркиваю теоретически. Однако, как уже упоминалось, в реальной жизни все немного сложнее, чем в проектах. Во-первых, необходимо предоставлять по меньшей мере два файла в разных форматах видео: OGG и MP4. Причина заключается в том, что, хотя элемент `<video>` стандартизован, стандартного формата видео не существует. Одни браузеры поддерживают одну группу кодеков, другие — другую, и эти группы между собой могут совершенно не пересекаться. Во-вторых, у кодека, используемого в формате MP4 (единственном поддерживаемом такими важными браузерами, как Safari и Internet Explorer), коммерческая лицензия.

Форматы OGG и MP4 представляют собой контейнеры для видео и аудио. OGG включает в себя видеокодек Theora и аудиокодек Vorbis, а MP4 — видеокодек H.264 и аудиокодек AAC. Сейчас OGG поддерживают браузеры Firefox, Google Chrome и Opera, а MP4 работает в Safari, Internet Explorer и Google Chrome.

Элемент <video>

Давайте пока что постараемся позабыть о трудностях и насладиться простотой элемента `<video>`. У этого элемента несколько атрибутов, предназначенных для установки свойств и конфигурации по умолчанию. Атрибуты `width` и `height`, как и у любого другого элемента HTML,

объявляют размеры элемента `<video>` или окна проигрывателя. Размер самого видео автоматически подгоняется под эти значения, однако они не предназначены для растягивания картинки, поэтому их используют для ограничения области видеосодержимого и сохранения единообразия дизайна. Атрибут `src`, как говорилось ранее, позволяет указать источник видео. Этот атрибут можно заменить элементом `<source>` и его собственным атрибутом `src` и объявить несколько источников для видео в разных форматах (как в следующем примере).

Листинг 5.2. Работающий в разных браузерах видеопроигрыватель с элементами управления по умолчанию

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <title>Видеопроигрыватель</title>
</head>
<body>
<section id="player">
  <video id="media" width="720" height="400" controls>
    <source src="http://minkbooks.com/content/trailer.mp4">
    <source src="http://minkbooks.com/content/trailer.ogg">
  </video>
</section>
</body>
</html>
```

В листинге 5.2 мы расширили определение элемента `<video>`, и теперь между его тегами находятся два элемента `<source>`. Они определяют разные источники видео, для того чтобы браузер мог сделать правильный выбор. Браузер считывает теги `<source>` и, основываясь на поддерживаемых форматах (MP4 или OGG), принимает решение, какой из файлов следует воспроизвести.

САМОСТОЯТЕЛЬНО

Создайте новый пустой HTML-файл с именем video.html (или любым другим), скопируйте в него код из листинга 5.2 и откройте в разных браузерах, чтобы проверить функционирование элемента <video>.

Атрибуты элемента <video>

Вы наверняка заметили в теге `<video>` один атрибут, который появился как в листинге 5.1, так и в листинге 5.2. Атрибут `controls` — один из нескольких специфических атрибутов, доступных для данного элемента. Он отображает элементы управления видео, предоставляемые самим браузером. Каждый браузер активирует собственный интерфейс, при помощи которого пользователь может начать воспроизведение видео, приостановить его, перейти к определенному кадру и т. п.

Помимо `controls` можно использовать также следующие атрибуты:

- `autoplay`. Когда этот атрибут присутствует, браузер автоматически запускает воспроизведение видео, как только это становится возможным;
- `loop`. Если этот атрибут присутствует, браузер начинает воспроизведение видео с начала, как только ролик достигает конца;
- `poster`. Этот атрибут позволяет указать URL-адрес изображения, которое будет отображаться на экране, пока воспроизведение видео не запущено;
- `preload`. Этот атрибут может принимать три значения: `none`, `metadata` и `auto`. Первое указывает, что видео не должно кэшироваться и обычно используется для минимизации расходования трафика. Второе значение, `metadata`, рекомендует браузеру загрузить определенную информацию о ресурсе, например ширину и высоту картинки, продолжительность, первый кадр. Третье значение, `auto`, устанавливается по умолчанию и заставляет браузер загружать файл целиком, как только это становится возможным.

Листинг 5.3. Использование атрибутов тега <video>

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <title>Видеопроигрыватель</title>
</head>
<body>
  <section id="player">
    <video id="media" width="720" height="400" preload controls
      loop poster="http://minkbooks.com/content/poster.jpg">
      <source src="http://minkbooks.com/content/trailer.mp4">
      <source src="http://minkbooks.com/content/trailer.ogg">
```

```
    </video>
  </section>
</body>
</html>
```

В листинге 5.3 мы заполнили элемент `<video>` атрибутами. Из-за того что поведение элемента в разных браузерах может различаться, некоторые атрибуты будут по умолчанию включены или выключены, а часть в зависимости от обстоятельств вообще не будет работать. Для того чтобы получить полный контроль над элементом `<video>` и воспроизводимым видеороликом, необходимо запрограммировать собственный видеопроигрыватель на JavaScript, пользуясь возможностями новых методов, свойств и событий, появившихся в спецификации HTML5.

Программирование видеопроигрывателя

Если вы тестировали предыдущие примеры кода в разных браузерах, то наверняка обратили внимание на то, что в каждом из них используется собственный графический дизайн элементов управления воспроизведением. В каждом браузере собственные кнопки и индикатор прогресса, даже собственный набор функций. Такое положение дел в определенных ситуациях может нас устраивать, однако в профессиональной среде, где каждая деталь имеет значение, единообразие дизайна обязательно должно сохраняться во всех механизмах и приложениях, и мы должны полностью контролировать процесс от начала и до конца.

Спецификация HTML5 предоставляет новые события, свойства и методы манипулирования видео и интегрирования его в документы. Теперь мы можем создавать собственные видеопроигрыватели и наполнять их желаемой функциональностью, применяя для этого HTML, CSS и JavaScript. Теперь видео стало частью документа.

Дизайн

Каждому видеопроигрывателю требуется панель управления, предоставляющая хотя бы базовые возможности. В новом шаблоне в листинге 5.4 после элемента `<video>` мы добавили новый элемент `<nav>`. Элемент `<nav>` включает в себя два элемента `<div>`, `buttons` и `bar`, представляющих кнопку Play и индикатор прогресса.

Листинг 5.4. HTML-шаблон для нашего видеопроигрывателя

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <title>Видеопроигрыватель</title>
  <link rel="stylesheet" href="player.css">
  <script src="player.js"></script>
</head>
<body>
<section id="player">
  <video id="media" width="720" height="400">
    <source src="http://minkbooks.com/content/trailer.mp4">
    <source src="http://minkbooks.com/content/trailer.ogg">
  </video>
  <nav>
    <div id="buttons">
      <button type="button" id="play">Play</button>
    </div>
    <div id="bar">
      <div id="progress"></div>
    </div>
    <div style="clear: both"></div>
  </nav>
</section>
</body>
</html>
```

Шаблон также включает в себя две ссылки на внешние файлы с кодом. Один из файлов — это player.css, содержащий CSS-стили, представленные в листинге 5.5.

Листинг 5.5. CSS-стили для проигрывателя

```
body{
  text-align: center;
}
header, section, footer, aside, nav, article, figure, figcaption,
hgroup{
  display : block;
}
#player{
  width: 720px;
```

```
  margin: 20px auto;
  padding: 5px;
  background: #999999;
  border: 1px solid #666666;

  -moz-border-radius: 5px;
  -webkit-border-radius: 5px;
  border-radius: 5px;
}
nav{
  margin: 5px 0px;
}
#buttons{
  float: left;
  width: 85px;
  height: 20px;
}
#bar{
  position: relative;
  float: left;
  width: 600px;
  height: 16px;
  padding: 2px;
  border: 1px solid #CCCCCC;
  background: #EEEEEE;
}
#progress{
  position: absolute;
  width: 0px;
  height: 16px;
  background: rgba(0,0,150,.2);
}
```

В коде листинга 5.5 для создания блока, который будет включать в себя все составляющие видеопроигрывателя, мы применили техники традиционной блочной модели. С помощью возможностей данной модели блок выравнивается по центру окна. Обратите внимание на третий тег `<div>` в конце элемента `<nav>` нашего шаблона. Он содержит строчный стиль, восстанавливающий нормальный поток документа.

В последнем примере кода нет никаких новых свойств и прочих сюрпризов — это всего лишь группа свойств CSS (которые вы уже изучили), описывающих стили элементов проигрывателя. Однако один стиль все же

можно назвать необычным: атрибут `width` для элемента `<div> progress` инициализируется со значением 0. Это необходимо потому, что мы будем использовать данный элемент для имитации индикатора прогресса, который меняется по мере воспроизведения видео.

САМОСТОЯТЕЛЬНО

Скопируйте новый шаблон из листинга 5.4 в HTML-файл (video.html). Создайте два новых пустых файла для CSS-стилей и JavaScript-кода и присвойте им названия player.css и player.js соответственно. Скопируйте код из листинга 5.5 в CSS-файл, а все следующие фрагменты кода JavaScript — в JavaScript-файл.

Код

Теперь настало время написать код JavaScript для проигрывателя. Запрограммировать видеопроигрыватель можно разными способами, однако мы продемонстрируем применение только самых необходимых событий, методов и свойств, с помощью которых обрабатывается видео. Остальное можете изучить самостоятельно и реализовать любые усовершенствования, которые подскажет воображение.

Мы будем работать с несколькими простыми функциями для запуска и приостановки воспроизведения видео, отображения индикатора прогресса воспроизведения, а также реализации возможности щелкнуть на индикаторе и перейти к соответствующему кадру вперед или назад.

События

В HTML5 появились новые события, относящиеся к конкретным API. Для обработки видео и аудио были добавлены события, информирующие о состоянии мультимедиа — например, о том, какая доля видео уже загружена, завершилось ли воспроизведение файла, воспроизводится видео или приостановлено и т. п. Мы не будем использовать в примере все существующие события, однако для построения более сложных приложений они могут понадобиться. Далее перечислены наиболее часто используемые:

- `progress`. Это событие срабатывает периодически и обновляет информацию о состоянии загрузки мультимедиа. Для доступа к этой инфор-

мации используется атрибут `buffered`, с которым мы познакомимся чуть позже;

- `canplaythrough`. Это событие срабатывает, когда становится известно, что весь файл мультимедиа может быть воспроизведен без перерывов. Статус устанавливается с учетом текущей скорости загрузки и в предположении, что она сохранится до завершения загрузки файла. Существует и другое событие, выполняющее аналогичную функцию, — `canplay`. Однако оно не учитывает ситуацию в целом и срабатывает, когда доступными становятся всего пара кадров;
- `ended`. Срабатывает, когда мультимедиа воспроизводится до конца;
- `pause`. Срабатывает, когда воспроизведение приостанавливается;
- `play`. Срабатывает при запуске воспроизведения мультимедиа;
- `error`. Срабатывает, когда происходит ошибка. Это событие доставляется в элемент `<source>`, соответствующий источнику мультимедиа, на котором произошла ошибка.

Посредством нашего проигрывателя будем прослушивать только обычные события `click` и `load`.

ВНИМАНИЕ

События, методы и свойства для API-интерфейсов пока что находятся на этапе разработки. В этой книге мы изучаем только те из них, которые важны для выполнения задачи и без которых создать приложение попросту не получится. Для того чтобы проверить, каких успехов достигла спецификация HTML5 в этом отношении, посетите наш веб-сайт и пройдитесь по ссылкам для этой главы.

Листинг 5.6. Первая функция

```
function initiate() {
  maxim=600;
  mmedia=document.getElementById('media');
  play=document.getElementById('play');
  bar=document.getElementById('bar');
  progress=document.getElementById('progress');

  play.addEventListener('click', push, false);
  bar.addEventListener('click', move, false);
}
```

В листинге 5.6 представлена первая функция для нашего видеопроигрывателя. Функция называется `initiate`, так как она инициирует приложение, то есть запускает его выполнение после завершения загрузки окна.

Так как эта функция исполняется первой, необходимо установить глобальные переменные для настройки проигрывателя. Используя селектор `getElementById`, мы определили ссылки на все элементы проигрывателя, для того чтобы позднее обращаться к ним в коде. Мы также установили переменную `maxim`, чтобы всегда знать максимальный размер индикатора прогресса (600 пикселов).

Наш проигрыватель должен принимать во внимание два действия: пользователь щелкает на кнопке Play (Воспроизведение) и пользователь щелкает на индикаторе прогресса, для того чтобы перейти к кадру ближе к началу или к концу медиафайла. Для этой цели мы добавили два прослушивателя событий. Сначала добавили к элементу `play` прослушиватель события `click`. Этот прослушиватель будет запускать функцию `push()` каждый раз, когда пользователь щелкнет на элементе (кнопке Play). Второй прослушиватель относится к элементу `bar`. Он указывает, что каждый раз, когда пользователь щелкает на индикаторе прогресса, будет выполняться функция `move()`.

Методы

Функция `push()`, код которой представлен в листинге 5.7, — это первая из функций, выполняющих конкретные действия. В зависимости от ситуации она вызывает специальные методы `pause()` и `play()`.

Листинг 5.7. Эта функция запускает и приостанавливает воспроизведение видео

```
function push(){
  if(!mmedia.paused && !mmedia.ended) {
    mmedia.pause();
    play.innerHTML='Play';
    window.clearInterval(loop);
  }else{
    mmedia.play();
    play.innerHTML='Pause';
    loop=setInterval(status, 1000);
  }
}
```

Специальные методы `play()` и `pause()` входят в список методов, добавленных в HTML5 специально для обработки мультимедиа. Далее перечислены наиболее часто используемые:

- `play()`. Запускает воспроизведение медиафайла с самого начала, если только он не воспроизводился до этого и не был приостановлен;
- `pause()`. Приостанавливает воспроизведение;
- `load()`. Загружает медиафайл. Его полезно применять для того, чтобы в динамических приложениях загружать мультимедиа заранее;
- `canPlayType(тип)`. Благодаря этому методу мы узнаем, поддерживается ли определенный формат файла браузером.

Свойства

В функции `push()` также используются некоторые свойства, позволяющие получать информацию о мультимедиа. Далее перечислены наиболее популярные:

- `paused`. Возвращает значение `true`, если воспроизведение мультимедиа приостановлено или еще не начиналось;
- `ended`. Возвращает значение `true`, если воспроизведение мультимедиа завершилось;
- `duration`. Возвращает длительность мультимедиа в секундах;
- `currentTime`. Способно получать и возвращать значение. Оно либо информирует о текущей позиции воспроизведения мультимедиа, либо устанавливает новую позицию воспроизведения;
- `error`. Возвращает значение ошибки;
- `buffered`. Предоставляет информацию о том, какая доля файла уже загружена в буфер. Благодаря этому мы получаем возможность создавать индикаторы прогресса загрузки. Обычно данное свойство считывают после срабатывания события `progress`. Поскольку пользователи могут запрашивать загрузку мультимедиа не только с начала, но и с любой другой позиции, свойство `buffered` возвращает массив, содержащий все уже загруженные части мультимедиа, а не только ту, начало которой совпадает с началом файла. Для обращения к элементам массива используются атрибуты `end()` и `start()`. Например, код `buffered.end(0)` возвращает продолжительность (в секундах) первой из содержащихся в буфере частей мультимедиа. Поддержка данной возможности еще окончательно не реализована.